uDIREC: Unified Diagnosis and Reconfiguration for Frugal Bypass of NoC faults

Ritesh Parikh
University of Michigan
Ann Arbor, MI - 48109
parikh@umich.edu

Valeria Bertacco
University of Michigan
Ann Arbor, MI - 48109
valeria@umich.edu

ABSTRACT

As silicon continues to scale, transistor reliability is becoming a major concern. At the same time, increasing transistor counts are causing a rapid shift towards large chip multi-processors (CMP) and system-on-chip (SoC) designs, comprising several cores and IPs communicating via a network-on-chip (NoC). As the sole medium of on-chip communication, a NoC should gracefully tolerate many permanent faults.

We propose uDIREC, a unified framework for permanent fault diagnosis and subsequent reconfiguration in NoCs that provides graceful performance degradation with increasing number of faults. Upon in-field transistor failures, uDIREC leverages a fine-resolution diagnosis mechanism to disable faulty components very sparingly. At its core, uDIREC employs a novel routing algorithm to find reliable and deadlock-free routes that utilize the still-functional links in the NoC. uDIREC places no restriction on topology, router architecture and number and location of faults. Experimental results show that uDIREC, implemented in a 64-node NoC, drops 3× fewer nodes and provides 25% higher throughput (beyond 15 faults) when compared to other state-of-the-art fault-tolerance solutions. uDIREC's improvement over prior-art grows with more faults, making it a suitable NoC reliability solution for a wide range of fault rates.

Categories and Subject Descriptors

B.8.1 [**Performance and Reliability**]: Reliability, Testing and Fault Tolerance—*network-on-chip*; C.2 [**Computer-Communication Networks**]: Miscellaneous—*on-chip networks*

General Terms

Reliability, Algorithms

Keywords

NoC, permanent faults, diagnosis, reconfiguration

1. INTRODUCTION

To benefit from the continued technology scaling in microprocessors, recent trends are pointing towards a growing number of simpler, power-efficient cores-on-chip. As a result, the corresponding increase in inter-core communication demands has rapidly sidelined traditional interconnects, such as simple buses, due to their limited bandwidth and poor scalability. Networks-on-chip (NoCs), characterized by highly concurrent communication and better scalability, are becoming the *de facto* choice for on-chip interconnects. However, as the sole medium for on-chip communication, a NoC might also become a single point of failure. Moreover, technology experts predict frequent failures in the field at future technology nodes [7, 48]. Even for existing silicon, large scale studies have shown error rates that are orders of magnitude higher than previously assumed [33]. This waning reliability of silicon further exacerbates the problem.

Circuit techniques alone are not sufficient to tackle this reliability challenge, and reliability aware architectural and software solutions are becoming more important [43]. Researchers have already proposed future CMP architectures that can gracefully tolerate up to a few hundred (∼500) processor-logic permanent faults [24, 37] in a 64-node CMP. However, for extended chip lifetime, comparable fault-tolerant solutions are required for both processors and their interconnection fabric. In other words, reliability solutions should minimize the damage caused by faults in the interconnection network: these faults can potentially lead to a disconnected network and to the loss of healthy processing elements. State-of-the-art permanent fault-tolerant solutions for NoCs (fault-diagnosis: [17], reconfiguration: [1, 41]) fall significantly short of this goal, dropping the majority of potentially-healthy nodes at a high number of faults.

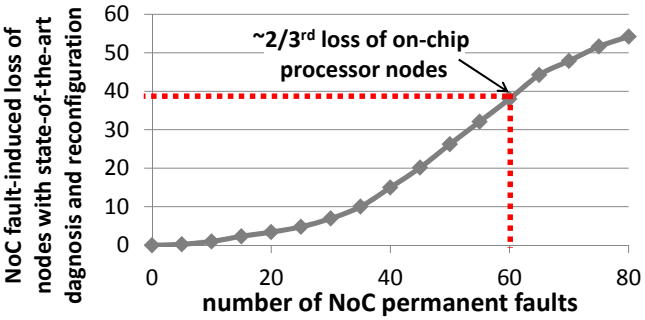


Figure 1: Loss of processing capability induced by faults in the NoC for a 64 node mesh network equipped with the state-of-the-art diagnosis [17] and reconfiguration [1, 41] schemes. Approximately 2/3*rd* of the nodes are unreachable at only 60 NoC faults.

We back our claim that current NoC fault-tolerance techniques are insufficiently robust with a quantitative study. Industrial CMP designs [49, 50], dedicate 6-15% of the chip area to the NoC and

MICRO -46, December 07-11 2013, Davis, CA, USA

ACM 978-1-4503-2638-4/13/12 ...$15.00

roughly 50% to the processor logic, while the rest is dedicated to memory, I/O, *etc.* Assuming uniform distribution of faults over silicon area, the NoC should be able to gracefully tolerate up to 60-150 faults to match processor-oriented reliability schemes [24, 37]. To analyze the effect of faults, we modeled a 64-node NoC, equipped with state-of-the-art fault-diagnosis [17] and route-reconfiguration schemes [1, 17]. Figure 1 plots the number of processing nodes unreachable by such a NoC with increasing faults. This figure shows that existing state-of-the-art solutions deteriorate considerably beyond 30 transistor faults, dropping many nodes with just a few additional faults (see slope change). At 60 NoC-related faults, almost 2/3*rd* of the nodes become isolated, many of which are expected to be functional cores.

To address this problem, we propose a novel solution, called uDIREC, which drops over 3× fewer nodes than existing solutions, and thus minimizes the network-induced loss of processing capability. Further, it is often the case with uDIREC that cores (and other on-chip functionalities) are only discarded when the number of faults is beyond the capacity of the fault-tolerant mechanisms protecting the cores themselves. uDIREC is beneficial for a variety of multi-core configurations (few or many cores) and varied fault rates (few to fault-ridden). For example, configurations with few cores degrade drastically and often become dysfunctional with the loss of one (or few) core(s). uDIREC extends the lifetime of such configurations by dropping the first (few) node(s) only after a significantly higher number of faults have manifested.

Existing NoC reliability solutions can be broadly divided into *architectural* protection against faults in the router logic [11, 17, 26] and *route-reconfiguration* solutions to bypass faulty links or routers [1, 8, 17, 28, 41]. Solutions in the latter category may serve as an additional reliability net, when router components are beyond the repair capability of the former category. When faults leads to a complete router loss, the failure is modeled by marking all the links connected to the affected router as faulty, and then re-routing around them. In this paper, we focus on reconfiguration solutions, as they provide a generic scheme to overcome all types of NoC faults.

Fine-resolution diagnosis of permanent faults is essential to minimize functionality- or processing- loss after reconfiguration around faulty components. Fortunately, a majority of a router's logic faults affects only small localized portions of the router [17]. To accurately assess this phenomenon, we injected stuck-at faults in a 5-port wormhole router gate-level netlist, with a spatial distribution proportional to the silicon area of gates and wires. Our analysis of the fault sites revealed that indeed most (96%) faults affect only small fractions of the router logic: their functionality can be entirely masked by disabling and re-routing around a single unidirectional router link. Assuming a mechanism capable of such fine-resolution diagnosis, an efficient reconfiguration scheme can potentially shield against many NoC faults, disabling only a single unidirectional link for each fault. However, existing on-chip, topology-agnostic route-reconfiguration solutions [1, 41] fail to exploit this opportunity as they can operate with bidirectional links only. For example, both Ariadne [1] and Immunet [41] make the network as an *undirected* graph, and are based on the construction of an *undirected* spanning tree on each reconfiguration event. Subsequently discovered routes are therefore valid only when each edge in the graph provides bidirectional connectivity. Therefore, these reconfiguration schemes unnecessarily consider a fault in one direction to be fatal for the entire bidirectional link, and *cannot* benefit from the fine-grained diagnosis information. Further, trivially modifying these algorithms does not allow the use of the healthy unidirectional links. As a result, a completely redesigned route-reconfiguration solution is required that is capable of separately exploring the routes away and towards the root node of the spanning tree, while still providing deadlock freedom.

1.1 Contributions

Existing approaches for building fault-tolerant NoCs disable whole routers and links to recover the communication fabric from permanent faults. This heavy-handed approach to recovery cannot tolerate the degree of faults predicted for future silicon. This work, uDIREC, presents a novel, frugal approach to reliable NoC design, pairing a fine-resolution diagnosis and careful shutdown with a capable adaptive routing algorithm, thus resulting in a much more graceful degradation. As transistors fail over time, uDIREC disables components frugally, so as to maintain the maximum working set of links, leading to high reliability and performance benefits. uDIREC's contributions are summarized below:

i) A new fine-grain fault model for NoCs that enables the frugal bypass of faulty unidirectional links, without disabling other healthy unidirectional link(s). We further develop a **low-cost diagnosis scheme**, inspired by [47], that delivers the fault locations adhering to our fine-grain fault model.

ii) A novel routing algorithm to maximally utilize unidirectional links in fault-ridden irregular networks that result from the application of our fine-grain fault model to faulty NoCs. The routing algorithm is guaranteed to discover only deadlock-free routes without requiring additional VCs. To the best of our knowledge, it is the first deadlock-free routing algorithm that utilizes unidirectional links and is topology-agnostic at the same time.

iii) A software-based reconfiguration scheme, which in collaboration with our diagnosis scheme, handles in-field permanent faults in NoCs. It places no restriction on topology, router architecture or the number and location of faults. Internally, it utilizes our novel routing algorithm to discover a new set of deadlock-free routes for the surviving topology.

iv) uDIREC (for **u**nified **DI**agnosis and **REC**onfiguration). The integration of our fine-grained diagnosis and reconfiguration schemes enables a low-cost and frugally-degrading reliability solution. Experiments on a 64-node NoC with 10-60 interconnect-related faults show that uDIREC drops 60-75% fewer nodes and provides 14-40% higher throughput over other state-of-the-art fault-tolerance solutions.

The rest of this paper is organized as follows: Section 2 introduces our fine-grain fault model and diagnosis scheme. Our novel deadlock-free routing algorithm is detailed in Section 3, while the reconfiguration scheme is described in Section 4. Sections 5 presents our experimental results, while Sections 6 and 7 present related work and conclusions.

2. FAULT MODEL AND DIAGNOSIS

In this section, we discuss the unique characteristics of fault manifestation in NoCs and the opportunities they present for improved reliability and performance. Routers in a NoC are typically connected by **full-duplex bidirectional links**, with each link direction using a separate set of wires [5, 49, 50]. Given this property, we can significantly increase the robustness of the system by letting a single fault only impair one direction of one link. How-

ever, most previous reconfiguration approaches assume a very simple fault model, where a faulty wire in one direction is tagged as a failure of the entire bidirectional link. Any fault in the router logic is modeled by tagging all links connected to it as faulty. Vicis [17] first improved on this simplistic fault model by mapping gate-level faults within a router logic to specific *bidirectional* link failures. We call this coarse-grain fault model *Coarse_FM*: it constrains the residual network to have bidirectional links only. Reconfiguration solutions based on the *Coarse_FM* [1,17,41] are therefore often inspired by irregular routing algorithms designed for networks with bidirectional links only [10,46]. Up*/down* routing [46] is a classic example of a topology-agnostic algorithm for networks with bidirectional links that was adopted for providing fault-tolerance via route-reconfiguration [1]. Up*/down* works by assigning directions to all links in the network: *up* or *down*. Links towards the root node (connecting to a node closer to the root) are tagged as *up* links, while links away from the root are tagged as *down* links. Links between nodes equidistant to the root are tagged arbitrarily. All cyclic dependencies are broken by disallowing routes traversing a *down* link followed by an *up* link.

With this work, we propose a novel and refined fault model where all link- or router- level faults are mapped to *unidirectional* links. For example, transistor faults in an output buffer, in link wires or at an input FIFO are modeled as failures of the corresponding unidirectional link, as described in the next section. Faults fatal to the entire router operation, are modeled by tagging all links connected to the router as faulty. We call this fine-grain fault model *Fine_FM*. uDIREC leverages the additional links reported as fault-free by the *Fine_FM* to improve the reliability and performance of faulty networks.

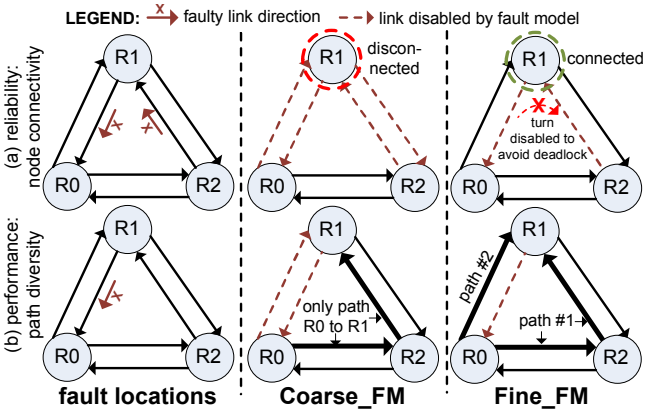


Figure 2: *Fine_FM* **provides additional working unidirectional links**. They can be utilized to (a) connect more nodes (reliability), and (b) provide path diversity (performance).

Improved Reliability: A solution based on the *Fine_FM* can provide better connectivity than the *Coarse_FM*, as the number of working links shrinks faster for the latter. In Figure 2a, *R1* is isolated when using the *Coarse_FM*. However, the *Fine_FM* allows deadlock-free routes between any pair of nodes by simply disabling the turn *R0*→*R1*→*R2*.

Improved Performance: The *Fine_FM* can improve path diversity over the *Coarse_FM*, as working unidirectional links can be used to transmit packets. In Figure 2b, the unidirectional link *R1*→*R0* is faulty. With the *Coarse_FM*, only one path remains from *R0* to *R1*, via links *R0*→*R2* and *R2*→*R1*. However, with the *Fine_FM*, an additional route can be utilized from *R0* to *R1*, via the unidirectional link *R0*→*R1*.

2.1 Fine-Resolution Fault Diagnosis

NoC faults, though most can be localized to single unidirectional links, are difficult to diagnose at fine-granularity. Vicis [17] proposed embedding a BIST unit at each router for this purpose, similar to [12]. However, its area overhead of 10% (reported in [16]) makes such a solution expensive. End-to-end directed, traffic-based diagnosis solutions [42] have also been proposed to lower the area overhead, however, they too stall the network operation while testing. Although low-cost assertion checkers for fault detection have recently been proposed [39], they do not provide diagnostic information about the faulty component(s). Fortunately, a low-cost, passive, fine-grained scheme for detection and diagnosis of NoC link faults was proposed in [47]. Their approach leverages a software-implemented centralized scoreboard that maintains the probability of each NoC link being faulty. The scoreboard is updated by passively monitoring traffic delivered at each network node. [47] also adds flit-level end-to-end ECC to detect and correct data faults in the transmitted packet. Upon an erroneous transmission, the work in [47] uses the ECC information to correct the packet and extract its source and destination addresses. This information is then sent to a designated "supervisor node" responsible for maintaining the scoreboard. The "supervisor node" updates all scoreboard entries corresponding to the links in the routing path of the erroneous packet (determined by analyzing the source and destination node addresses), to track the likelihood of any of these links being faulty. After accumulating the fault probabilities from just a few erroneous packets (15), the scoreboard entry with the highest accumulated fault probability is declared faulty. The resulting diagnosis was reported to be accurate more than 98% of the time, with the accuracy increasing over more monitored packets. Note that [47] reports that this solution is successful both on deterministic and minimally adaptive routing algorithms. However, this diagnosis scheme is limited to faults affecting the links between the routers and it does not tackle faults within the router logic.

The work in [47] was extended by [20], where each fault is mapped at the architectural level to a link(s) failure. Consequently, [20] can localize fault manifestations in any router component (links, datapath or control logic). This solution achieves the same high accuracy of diagnosis as [47], at an area cost of less than 3% [20], and without introducing any test-specific traffic into the NoC. The extension was based on two innovations: i) the deployment of simple counters and timers to diagnose errors affecting the router's control logic at fine-granularity, and ii) the perception of the router's datapath as an extension of the links connected to it. Elaborating on the latter, faults in the following set of datapath components are not differentiable from a route-reconfiguration perspective:

- output port buffer at the upstream router,
- link between routers,
- input port buffer at the downstream router, and
- crossbar contacts to (from) the output (input) port.

Thus, faults in all these components can be modeled as a single link failure, and handled by rerouting around that link. We call the combined set of these datapath components, a *datapath segment*, while we refer to datapath segments corresponding to opposite unidirectional links as a *datapath segment pair*. In this work, we further the diagnosis resolution by recognizing that the two opposite unidirectional links between adjacent routers, are associated with

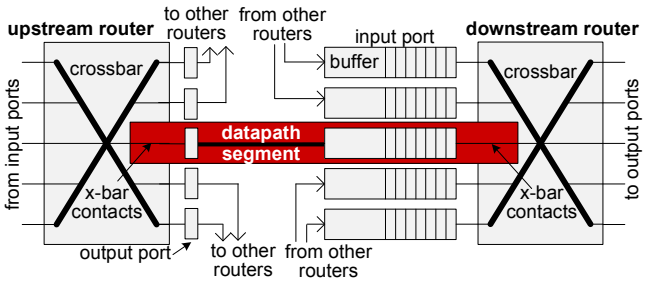


Figure 3: **A datapath segment** includes the crossbar contacts to the output port and the output port in the upstream router, the link, the input port buffer and the crossbar contacts from it in the downstream router.

independent datapath segments, and faults in one datapath segment do not affect the functionality of the other. We empirically studied the micro-architecture of wormhole routers to establish that a majority of faults can indeed be masked by re-routing around a single datapath segment. To this end, we synthesized a 5-port baseline mesh router, and conceptually divided all router components into two pools: i) components that only affect one datapath segment's functionality, and ii) components that affect the entire router's functionality. Examples of former category are crossbars, output ports, input ports and links, while the latter category includes arbiters, allocators and routing table. Components that only affect one datapath-segment, accounted for 96% of the router's silicon area: 96% of the faults will affect only one datapath segment, assuming an area-uniform fault distribution. We ascertained our observations by randomly injecting stuck-at faults at gate outputs of every such component, and testing the "unaffected" datapath segments for proper functionality with random test vectors.

Finally, to decouple the fault diagnosis for two segments in a pair, we extend the fault-probability scoreboard to maintain the fault probabilities for each datapath segment separately. With this addition, any router datapath fault can always be localized to a single datapath segment, and hence can be overcome by disabling a single unidirectional link, as illustrated in Figure 3. Since the majority of a router's area is dedicated to datapath components, most NoC faults can be masked by disabling only single unidirectional links. For simplicity of presentation, we will predominantly use "unidirectional link" to refer to a datapath segment. With our modifications, the *Fine_FM* can be adopted with only minimal overhead, providing very graceful performance degradation with increasing faults. For experiments, we have modeled our extension of [20], which can localize most fault manifestations to the resolution of a single datapath segment. The same diagnosis information is provided to all evaluated route-reconfiguration schemes (uDIREC and prior works). However, prior works such as Ariadne [1] and Immunet [41], cannot utilize this fine-grained fault localization information due to the limitations of their underlying routing algorithms.

3. ROUTING ALGORITHM

The constraint that all network links are bidirectional enables a desirable property: if a path between two nodes exists, irregular routing algorithms based on spanning tree construction can enable at least one deadlock-free route between them. In contrast, finding deadlock-free routes between any pair of nodes in a connected network is not always possible if the network has unidirectional links. Since our routing algorithm must enable only deadlock-free routes, it is possible to have to sacrifice a few connected nodes to achieve this goal.

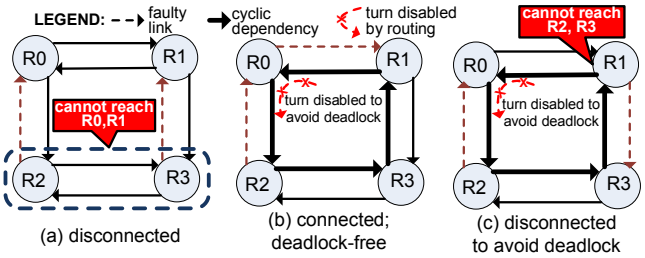


Figure 4: (a) Network disconnected. (b) Deadlock-free connectivity possible by disabling turn *R1→R0→R2*. (c) Network connected but deadlock freedom not possible without sacrificing connectivity.

3.1 Connectivity and Deadlock Freedom

We consider a network to be connected only if transmission is possible between any pair of nodes in either direction (two-way connectivity). Additionally, we assume that VCs are only used for separation of traffic into classes and/or avoiding deadlock in client protocols, but we do not require them to overcome routing deadlocks [23]. Based on their connectivity characteristics, we can divide all networks with unidirectional links into three categories:

(i) **Disconnected**: The network in Figure 4a is disconnected as traffic cannot traverse from *R2 & R3* to *R0 & R1*.

(ii) **Connected and deadlock-free**: In Figure 4b, a connected network has a cyclic dependency in the anticlockwise direction. Such a cycle can cause deadlock, as packets residing in routers' buffers can be indefinitely waiting for packets in front of them to free up buffer space. Formally, as a sufficient condition for deadlock freedom, all such dependency cycles should be eliminated [13]. Thus, we use an approach based on disabling certain connections between links, called 'turns' in [21], so that the packets cannot form a cyclic dependency. For example, in Figure 4b, the *R1→R0→R2* turn is disabled to break the anticlockwise cycle. In other words, the routing algorithm prohibits messages to go from *R1* to *R2 & R3* via *R0* to avoid packets from deadlocking. Connectivity is maintained even after disabling this turn.

(iii) **Disconnected to avoid deadlock**: The network in Figure 4c is connected as messages can be exchanged between any pair of nodes. However, if the turn *R1→R0→R2* is disabled to break the anticlockwise cycle, connectivity is lost. Specifically, *R1* cannot send messages to *R2 & R3*, as the only path connecting the concerned nodes goes through this disabled turn. Even when disabling any other turn to break the cyclic dependency, connectivity is jeopardized.

3.2 Deadlock-Free Routing Algorithm

Our routing algorithm is designed to maximally utilize resources in faulty networks that result from the application of our *Fine_FM*. In other words, it can be applied to discover deadlock-free routes in any irregular network with unidirectional links. uDIREC deploys our routing algorithm on each fault manifestation, to quickly discover reliable routes between the still-connected nodes. Our routing algorithm works by constructing two separate spanning trees with unidirectional links: one for connections moving traffic away from the root node (down-tree), and the other for connections moving traffic towards the root node (up-tree). Each node is then assigned a unique identifier corresponding to each tree: identifiers increase numerically with increasing distance from (to) the root in the down-tree (up-tree), while equidistant nodes are ordered arbi-

trarily. This leads to a unique ordering of nodes (lower order = closer to root) in each tree. Thereafter, the *up* link is defined as the unidirectional link towards the node with the lower identifier in the up-tree and the *down* link is defined as the unidirectional link towards the node with the lower identifier in the down-tree.

Lockstep Construction. The two spanning trees, however, cannot be constructed independently of each other. Because we use unidirectional links, such an approach could lead to a mismatch in the node ordering between the trees, and links could consequently receive inconsistent tags: *up* or *down*. An example of such situation is shown in Figure 5, where mismatched node orderings lead to link *R1→R2* being tagged *up* in the up-tree and *down* in the down-tree. Thus, the construction of the two trees must be in lockstep, guaranteeing matched ordering by construction.

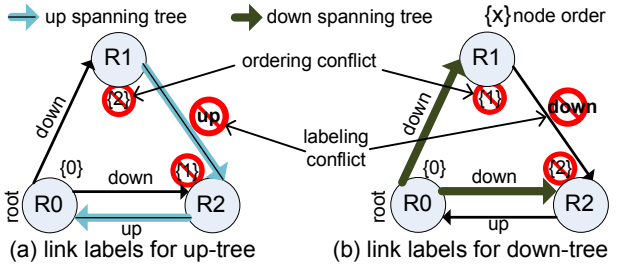


Figure 5: **Independent construction of up-tree and down-tree** causes inconsistent labeling of link *R1→R2*. (a) Up-tree: link is towards a node that is closer to the root; hence tagged *up*. (b) Down-tree: link is between nodes at the same level; hence tagged arbitrarily as *down*.

Matched-Ordering by Construction. Our routing algorithm builds the two trees using a breadth-first search, but advances the construction of the two trees in lockstep, expanding to new nodes only if a node order matching on both trees exists. To this end, each leaf node reached in the network expands the two trees to its descendants only when the node itself is reachable by both the up-tree and the down-tree. Otherwise, the up-tree (down-tree) construction is halted until both tree constructions reach that node. All nodes that are reachable by both the up-tree and the down-tree can communicate among themselves by enabling deadlock-free routes. All other unreachable nodes timeout after waiting for one or both tree(s) to reach them. As shown in Figure 7a, starting from the root node (*R0*), both the up-tree and the down-tree expand to *R2* using *bidirectional link R0↔R2*; hence *R2* can expand to its descendants. At the same time, the down-tree expands to *R1* and halts at *R1* for the up-tree to catch-up. In the next iteration, *R2* expands the up-tree to *R1*, cancelling the halting status of *R1*. At this time, both trees reach all nodes, and hence the network is connected and deadlock-free. In essence, our routing algorithm modifies the structure of the two spanning trees so as to achieve a consistent ordering of nodes. In practice, this construction algorithm may shuffle the order of nodes at the same level in one tree, so that the ordering matches that of the other tree. In the end, the two trees agree on node ordering and the links have been assigned consistent tags. Thereafter, all routes traversing a *down* link followed by an *up* link (*down→up* turn) are disallowed. Finally, a route search algorithm finds the minimal route(s) between each source-destination pair. A pseudo-code of the deadlock-free routing algorithm just described is provided in Figure 6. The algorithm exhibits the following properties:

Property 1: Our routing algorithm provides deadlock-free connectivity among all nodes reachable by both up-tree and down-tree.

```
ROOT = pick_root()
newly_reached = ROOT; node_order[ROOT] = 0
up-tree = down-tree = NULL;
/*Begin up-tree and down-tree construction*/
do:
| for(NODE in newly_reached):
| | up-tree += remaining_nodes_connected_to(NODE)
| | down-tree += remaining_nodes_connected_from(NODE)
| matched = overlap(up-tree,down-tree)
| up-tree -= matched; down-tree -= matched
| node_order = order_matched_nodes(matched)
| newly_reached = matched
while(newly_reached != NULL)
disable_unreached_nodes()
apply_down_up_turn_restrictions(node_order)
find_minimal_routes_with_turn_restrictions()
```

Figure 6: **Our deadlock-free routing algorithm** to determine deadlock-free routes in networks with unidirectional links. To guarantee a matched node ordering, nodes expand the trees to their neighbors only if both up-tree and down-tree have reached them. The resulting matched-ordering governs the turn restrictions, and the evaluated minimal routes adhere to these turn restrictions.

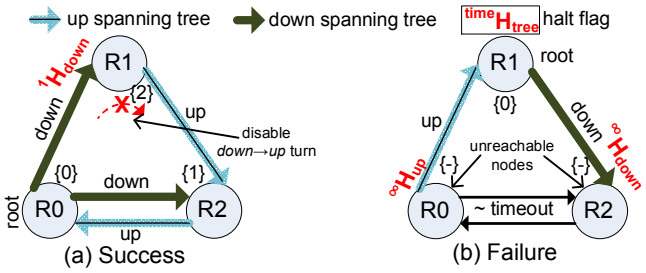


Figure 7: **Growing the up-tree and down-tree in lockstep.** The choice of root affects connectivity. (a) Success with root *R0*: both up-tree and down-tree connect all nodes with consistent labeling. (b) Failure with root *R1*: up-tree (down-tree) halted at *R0*(*R2*).

Proof's outline: With consistent node ordering, any deadlock causing cycle will contain at least one *up→down* turn and one *down→up* turn [13]. Since our routing algorithm guarantees consistent tagging of links, all dependency cycles are broken by disallowing just *down→up* turns. Additionally, all the nodes connected by the spanning trees can communicate deadlock-free, as any node can reach all other nodes by going to the root node first, thus following *up* links first and *down* links afterwards. In any such routing path, there will be no disabled *down→up* turn.

Property 2: Routing configurations produced by our routing algorithm perform at least as well as configurations generated by up*/down*, in any fault scenario.

Proof's outline: In the worst case, to provide consistent marking of links, both up-tree and down-tree can be build only via bidirectionally working links, as in up*/down*. Therefore, our routing algorithm is always able to connect at least as many nodes as up*/down* and is able to provide at least as many deadlock-free routes as up*/down*.

Root Node Selection. The structure of both trees greatly depends on the root node selection. However, as shown in Figure 7, this aspect may also affect the connectivity characteristics of the network. In this example, if instead of *R0* (Figure 7a), *R1* (Figure 7b) is chosen as root, our routing algorithm is unable to find deadlock-free routes to any other node in the network. With *R1* as root in Fig-

ure 7b, the up-tree uses the link *R0→R1* to expand to *R0* and the down-tree takes the link *R1→R2* to expand to *R2*. Both trees halt at their frontier nodes (*R0* for up-tree; *R2* for down-tree) waiting for their counterpart trees. The algorithm terminates with *R1* connected to no other node, as the down-tree (up-tree) never reaches *R0* (*R2*) in this configuration. Therefore, optimal root selection can improve the connectivity characteristics of the network when using our routing algorithm.

4. RECONFIGURATION

We designate any one node in the multi-core system as the "supervisor", which implements our reconfiguration algorithm in software. We take advantage of the fact that the fault diagnosis scheme we utilize already stores the topology information in a software-maintained scoreboard at any one node. For simplicity of implementation, we designate the same node as the "supervisor". With this setup, only the supervisor node is entitled to make diagnostic decisions on the health of the NoC, and therefore, information about fault locations and the surviving topology is already present at the supervisor. In this manner, we avoid hardware overhead incurred by software-based reconfiguration solutions [31, 46] to reliably collect the topology information at a central node. Upon a new fault detection, the supervisor node transmits a reserved message to all routers/nodes in the system, informing them about recovery initiation. On receiving this message, all routers suspend their current operation and wait for routing function updates from the supervisor, while the nodes stop new packet injections. In the meantime, the supervisor computes deadlock-free routes for the surviving topology, using the routing algorithm described in the previous section. In the naïve variant of our reconfiguration algorithm, the "supervisor" node is always chosen as the default root of the reconfiguration process. We call this version *(uDIREC_nv)*. The computed tables are finally conveyed back to each router/node, which then resume normal operation. Later in this section, we will describe area-efficient hardware structures used to reliably transmit the updated routing information from the supervisor node to all routers in the NoC.

As described in Section 3.2, network connectivity depends strongly on the choice of the root node, and therefore, the supervisor node may not always be the optimal root. To this end, our reconfiguration algorithm discovers the largest connected topology via an exhaustive search of the root node that maximizes network connectivity. Each node, in turn, is appointed as the temporary root node and the number of nodes it can connect is calculated. The optimal root-selection is finalized when one of the following two conditions occur: (i) a root node that provides deadlock-free connectivity among all nodes is found; or (ii) all nodes have been considered as root node. At the end of the selection, we determine the winner-root, *i.e.,* the node that could connect the maximum number of nodes when chosen as root. The diagnostic and reconfiguration routines from the current supervisor node, are also migrated to the new root node determined by our reconfiguration algorithm. This allows for a simple logic structure for transmission of diagnostic messages to the supervisor node. In contrast to the naïve implementation of uDIREC *(uDIREC_nv)*, this complete version, denoted simply as *uDIREC*, provides better reliability and performance characteristics, as we will show in Section 5. The pseudo-code of our reconfiguration algorithm is shown in Figure 8.

```
/* Root Selection by connectivity evaluation */
ROOT_win = -1; max_connectivity = 0
for(ROOT in all_nodes):
| connectivity = eval_uDIREC_connectivity(ROOT)
| if(connectivity == num_nodes):
| | ROOT_win = ROOT; break;
| if(connectivity > max_connectivity):
| | ROOT_win = ROOT;
| | max_connectivity = connectivity
/* Route Construction for winner root */
apply_uDIREC_routing_algo(ROOT_win)
```

Figure 8: **uDIREC's reconfiguration algorithm.** All nodes are tried as root, and the root that provides maximum connectivity, is chosen to build the new network. Within each root trial, our novel unidirectional routing algorithm is leveraged to determine deadlock-free routes and determine connectivity.

4.1 Discussion

Reconfiguration duration. We realize that considering all nodes as root is inefficient, and algorithmic optimization is possible, but we trade reconfiguration time for simplicity of the algorithm. This is because permanent faults (even when up to tens or hundreds) are not frequent enough for reconfiguration duration to affect overall performance. Tree based routing algorithms can be efficiently implemented in software, and typically take only hundreds of milliseconds to complete (∼170ms [46]). Even though we run multiple iterations of our tree-based routing algorithm, we expect the reconfiguration overhead to be within a few seconds at worst. Assuming an aggressive life-span of 2 years for high-end servers and consumer electronics, and 150 NoC faults (Section 1) in the worst case, a NoC would suffer 1 fault every 5 days. Therefore, an overhead of few seconds per fault manifestation is negligible.

Reconfiguration-induced deadlock. Reconfiguration can cause routing deadlocks even if both the initial (before fault manifestation) and final (after reconfiguration) routing functions are independently deadlock-free [29, 38]. We avoid such deadlocks by identifying the packets that request a turn that is illegal according to the updated routing function. These packets are then ejected to the network interface of the router in which they are buffered at the time of reconfiguration. After reconfiguration, these packets are re-injected into the network upon buffer availability. Other state-of-the-art reconfiguration techniques [1, 41] utilize a similar technique to overcome reconfiguration-induced deadlocks.

Early diagnosis. The diagnosis scheme that we employ, pinpoints the fault locations in the router datapath before the faults grow to fatal proportions. The first few faults in the router datapath are corrected by the end-to-end ECC in the process of obtaining diagnostic information about fault locations. This presents the opportunity to keep using the network (for sometime) even after the fault has been diagnosed, as the initial few faults in the router datapath are within the correction capacity of the ECC. The pre-emptive diagnosis enables us to salvage the processor and memory state of the about-to-be-disconnected nodes while the network is still connected. Traditionally, computer architects have relied on checkpointing support [40] for this purpose, while a recent research proposal [15] adds emergency links to this end. Using our technique, it is possible to greatly simplify this additional reliability-specific hardware. uDIREC does not guarantee the integrity of packets that are traversing the network after fault manifestation and before fault detection, and relies on orthogonal recovery schemes [2, 32, 35] for that. However, the property of early diagnosis can greatly reduce the likelihood of fatal data corruptions and reduce the reliance on such recovery schemes.

Optimal root and tree. The choice of root node also affects the network latency and throughput characteristics [44]. In addition,

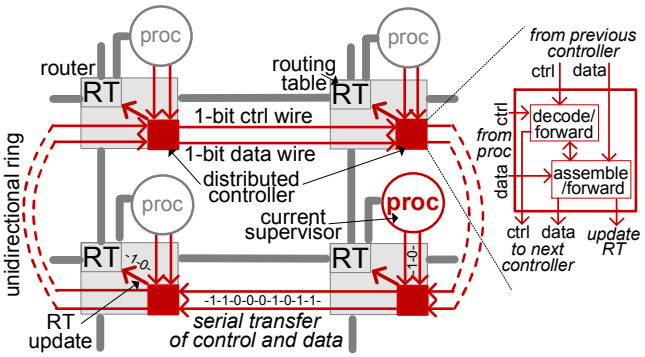


Figure 9: **Distribution of the routing tables from the supervisor.** One control and one data wire, organized in a unidirectional ring, are used to transfer routing tables to each NoC router in a serial fashion. Distributed controllers, one per router, snoop on the bit-stream broadcasted on the control wire and *decode* it to identify the data bit-stream relevant to the current router. The flagged bit-stream is then *assembled* and written into the routing table. As permanent faults are rare, this serial distribution poses insignificant overhead.

tree-based routing algorithms' performance is sensitive to the way trees are grown (breadth-first vs. depth-first), and the order in which nodes are numbered [44]. Further, the surviving set of on-chip functionalities may also differ with different root selection and tree growth schemes. However, the corresponding analysis is beyond the scope of this work and we do not consider these metrics in our root selection or tree-building process. uDIREC chooses the optimal node solely on the basis of number of connected nodes, breaking ties by comparing statically assigned node IDs, while the trees are build in a breadth-first fashion.

4.2 Routing Table Distribution

In a typical software-based reconfiguration scheme, such as the ones used for off-chip networks [45, 46], the surviving nodes first collaborate to choose the root node, following which, the surviving topology information is communicated to this root node. Finally, the newly computed routing tables are delivered to all the surviving nodes. Dedicated hardware resources are required to reliably communicate this information to and from the root node. Fortunately, uDIREC completely eliminates the need of collecting the NoC health status at a central node on each failure. The tight integration of the diagnosis and reconfiguration scheme in uDIREC makes it possible to utilize the topology information already available with the supervisor node.

We can drastically reduce the overhead of routing table distribution by noting that permanent faults are rare occurrences and all reconfiguration-specific transmissions can be done serially over a single wire. To this end, we utilize a combination of one control wire and one data wire, that are managed by distributed controllers (at each NoC router) organized in a ring topology. The control wire is utilized to notify all NoC routers of recovery initiation, and to set up the data communication between the supervisor and any particular router. Once the communication is set up, the data wire is used to transmit the routing table for that specific router, bit-by-bit. The process is repeated for all routers in the NoC: upon completion, an end-of-reconfiguration signal is broadcasted again via the same control wire. Specifically, a *decode* unit is included within each controller that snoops on the bit-stream broadcasted on the control wire and identifies data bit-streams relevant to the current router. Once a relevant data bit-stream is identified, the *assemble* unit gathers the information transmitted on the data wire and writes it to the routing table. We note that for a 64-node mesh NoC, the routing table at each node is 64 (destination nodes)$\times$4 (directions) bits. Thus, the routing information for the entire NoC is 2KB only, which should take insignificant time to transmit when compared to the 100s of milliseconds required for software-based route evaluation. The hardware implementation of the scheme just described is shown in Figure 9. The 2-bit wide links are used only during reconfiguration, and are otherwise disabled with power-gating, greatly reducing both the risk of wearout faults and the power overhead. Further, due to their small area footprint, simple resilience schemes to protect them (TMR or ECC), do not add significant overall overhead.

We synthesized our baseline 64-bit channel mesh router using Synopsys DC, targeting the Artisan 45nm library. We also estimated the link wire area for the same using Orion2.0 [25] considering dimensions from an industrial chip [49]. Our baseline router has a total area of 0.138mm^2 (logic=0.08mm^2,wire=0.06mm^2). In comparison, the distributed controller logic required to implement our routing table distribution scheme is trivial, and the two additional distribution wires (unidirectional ring) lead to an area overhead of only 0.34% compared to this baseline NoC. For this study, we assumed that the router wires and the two distribution wires are of the same length, and a unidirectional ring has 1/8th the number of channels compared to a mesh.

5. EXPERIMENTAL RESULTS

We evaluated uDIREC by modeling a NoC system in a cycle-accurate C++ simulator based on [14]. The baseline system is an 8x8 mesh network with a generic 4-stage pipeline (including link traversal) with 2-message classes, 1-VC per message class routers. Each input channel is 64-bits wide and each VC buffer is 8-entry deep. In addition, the NoC is augmented with uDIREC reconfiguration capabilities. All results are presented for both the naïve (uDIREC_nv) and complete (uDIREC) versions of our reliability scheme. Moreover, for comparison, we also implemented Ariadne [1], which outperformed all previous on-chip reconfiguration solutions, *i.e.*, Vicis [17] and Immunet [41]. Ariadne [1] reports 40% latency improvement over Immunet, which falls back to a high-latency ring to provide connectivity, and 140% improvement over Vicis, which occasionally deadlocks. Since both Ariadne and Immunet guarantee connectivity if routes (using only bidirectional links) between pairs of nodes survive, they show identical packet delivery rates. They also deliver a higher fraction of packets compared to Vicis, especially at high number of faults when Vicis tends to deadlock. Our results demonstrate uDIREC's substantial improvements over Ariadne, and thus uDIREC's improvements over Vicis and Immunet are expected to be even more impressive. Note that Ariadne assumed a perfect diagnosis mechanism, and therefore, for a fair comparison, we have paired it with the diagnosis scheme of [20] in our implementation.

Our framework is analyzed with two types of workloads (Table 1a): synthetic uniform random traffic, as well as applications from the PARSEC suite [4]. PARSEC traces are obtained from Wisconsin Multifacet GEMS simulator [30] modeling a fault-free network and configured as detailed in Table 1b. After fault injections, we ignore messages originating from and destined to the disconnected nodes, and this could lead to some evaluation inaccuracies for parallel collaborating benchmarks running on a partitioned multi-core. However, the traces are intended to subject the faulty NoC to the

realistic burstiness of application traffic, and provide a simple and intuitive comparison. The metrics provide valuable insights considering that a particular fault manifestation in uDIREC and prior work(s) could lead to vastly different configurations in terms of number/location/functionality of working cores/IPs.

(a)

traffic	uniform PARSEC
packet	1flit (control) 5flits (data)
simulation	1M cycles
warm-up	10K cycles

(b)

processor	in-order SPARC
coherence	MOESI
L1 cache	Private: 32KB/node ways:2 latency:3
L2 cache	Shared: 1MB/node ways:16 latency:15

Table 1: **(a) Simulation inputs. (b) GEMS configuration.**

Fault Injection. Our architectural-level fault injection technique randomly injects gate-level faults in network components with a uniform spatial distribution over their silicon area. Each fault location is then analysed to map it to dysfunctional link(s), modeling the fault diagnosis scheme proposed in [20]. The links that are marked as dysfunctional are bypassed using the route-reconfiguration schemes we evaluate. Further, our baseline NoC is not equipped with any reliability feature beside uDIREC. In our experiments, we injected a varying number of permanent faults (0-60 transistor failures) into the NoC infrastructure, and analyzed uDIREC's reliability and performance impact. For each number of transistor failures, the experiment was repeated 1,000 times with different fault spatial locations, selected based on a uniformly random function.

Scope of Experiments. We restricted the scope of our experiments by injecting faults only in the NoC infrastructure because the system-level performance and reliability depends on the vast number of unrelated factors: fault location, fault timing, memory organization, programming model, processor and memory reliability schemes, and architecture specific characteristics. As a result, the surviving system functionality might not be directly comparable. Hence, we have provided a generalized evaluation across a wide range of faults, consisting of insightful performance (latency, throughput) and reliability (number of dropped nodes, packet delivery rate) metrics for fault-tolerant NoCs.

5.1 Reliability Evaluation

As faults accumulate, networks may become disconnected. Performance of parallel workloads running on a multi-core chip with a faulty network directly depends on the number of connected processing elements (PEs) and other on-chip functionality, e.g., memory controllers, caches. Thus, the ability of an algorithm to maximize the connectivity of a faulty network is critical since, if no routes are available between two nodes, they cannot work collaboratively. In this section, we study: a) average number of dropped nodes, *i.e.*, nodes not part of the largest connected sub-network, and b) the packet delivery rate for uniform traffic, as faults accumulate in the NoC. Number of dropped nodes indicate the loss of on-chip processing elements and vital functionality, while the packet delivery rate captures the number of packets delivered over the number of packets generated. Both reflect the reliability of the network: a more reliable network will drop fewer nodes and will deliver a higher percentage of packets.

It can be noted in Figure 10 that Ariadne consistently drops over 3× more nodes than uDIREC. Even with just a few faults (5-20), uDIREC shows substantial improvement over Ariadne, dropping 1 node against Ariadne's 3 at 20 faults, as shown in the zoomed section of Figure 10. This showcases uDIREC's advantage across a

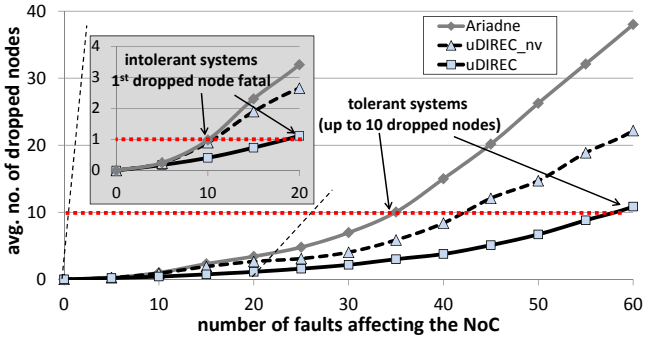


Figure 10: **Average number of dropped nodes**. Compared to Ariadne, uDIREC drops 3× fewer nodes and approximately doubles the number of faults tolerated before the same number of nodes are dropped. uDIREC_nv drops significantly fewer nodes compared to Ariadne at the same number of faults, but more than uDIREC.

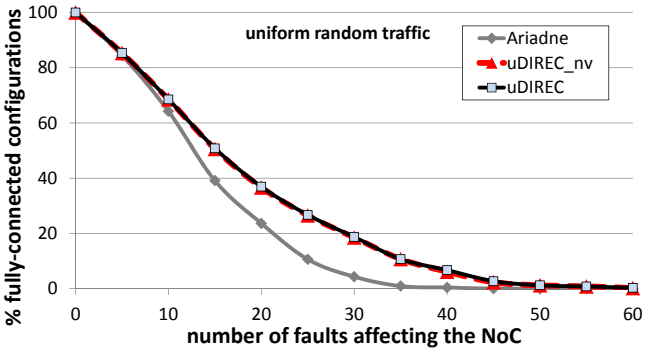


Figure 11: **Probability of completely connected configurations** decreases rapidly with increasing faults. uDIREC provides fully connected networks with a higher probability.

wide range of reliability requirements. For highly intolerant systems, where the average of one node loss is considered fatal, uDIREC can tolerate an average of 20 faults, as compared to Ariadne's 10 faults, as shown in the zoomed section of Figure 10. Figure 11 shows the probability of a completely connected network (when network connectivity is equivalent to a fault-free system) with a varying number of transistor faults. As can be noted from the figure, uDIREC consistently leads to more than 3× completely connected networks between 25 and 50 faults, when compared to Ariadne, beyond which the probability of still having a completely connected network is near-zero. Also, both uDIREC and uDIREC_nv show an almost similar probability of a completely connected network (overlapped lines in Figure 11). For more tolerant systems, for instance, those that can operate with an average loss of up to 10 nodes, uDIREC can keep the system functional up to 60 faults on average as compared to 35 faults in Ariadne's case. uDIREC_nv, a naïve variant of uDIREC, drops more nodes than uDIREC but still shows substantial improvement over Ariadne. With lower complexity and shorter reconfiguration, uDIREC_nv is a good trade-off for commodity systems.

A partitioned network is unable to deliver packets originating in a sub-network different from the destination sub-network. Analyzing Figure 12, both uDIREC and Ariadne deliver the majority (or all) of packets up to 10 faults. Beyond 15 faults, Ariadne starts partitioning into multiple sub-networks, and hence its delivery rate drops substantially below that of uDIREC. At 25 faults, uDIREC deliv-

ers 7% more packets than Ariadne, and the gain goes up to $\sim 3\times$ at 60 faults. uDIREC's ability to deliver a large fraction of packets even at a high number of faults makes it an excellent solution for fault-ridden NoCs. Again, uDIREC_nv delivers fewer packets than uDIREC, but considerably more packets than Ariadne. The number of dropped nodes in Immunet and its packet delivery rate, are both identical to Ariadne [1], whereas Vicis delivers a lower fraction of packets at higher number of faults. This clearly highlights the limitations of *Coarse_FM*, as both Ariadne and Immunet provide optimal connectivity considering the *Coarse_FM*.

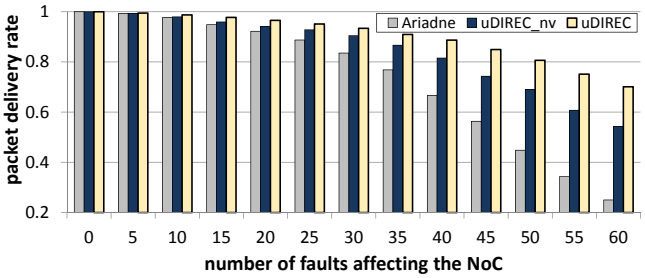


Figure 12: **Packet delivery rate**. Higher network partitioning in Ariadne causes steep decrease in delivery rate beyond 10 faults. In contrast, both uDIREC and uDIREC_nv degrade gracefully.

5.2 Performance Evaluation

After reconfiguration, the NoC should keep functioning adequately with only a graceful performance degradation. In our evaluation we report two performance metrics: average network latency and saturation throughput, after the network is affected by transistor faults. Latency and throughput measures are reported for the largest connected sub-network, assuming nodes disconnected from each other cannot work collaboratively. First, we report the average zero-load network latency, that is, the steady-state latency of a lightly loaded network (0.03 flits injected per cycle per node). It reflects the average delivery time of a network without congestion, and hence, in a sense, it indicates the average length of routes between NoC nodes. A reconfiguration technique that provides greater path diversity will have a lower zero-load latency.

Analyzing Figure 13, both uDIREC and Ariadne initially show an increase in latency because the number of paths affected increases with increasing faults, while very few nodes are disconnected from the network. Beyond approximately 30 faults, Ariadne's latency starts falling. This effect is easily understood by analysing Figure 10 beyond the 30-faults mark: a substantial number of nodes are dropped by Ariadne. As a result, packets now travel shorter routes to their destinations, and thus the average network latency is reduced.

uDIREC exhibits better latency characteristics when compared to Ariadne. Initially, uDIREC degrades gracefully as it is resource-conscious and provides greater path diversity. For example, at 30 faults, uDIREC on average has 7% lower latency than Ariadne. However, as more faults accumulate, Ariadne drops nodes more quickly than uDIREC (Section 5.1), and because of shorter routes in partitioned networks, Ariadne's latency shows a greater decrease. The crossover between the two latency graphs is at $\sim$40 faults, and as noted from the packet delivery rate chart copied over in Figure 13, the difference between the delivery rate of the two techniques is large (35% more in uDIREC) and grows rapidly beyond that point. uDIREC would show even bigger latency improvements over Immunet and Vicis, as they show 40% and 140% worse latency than

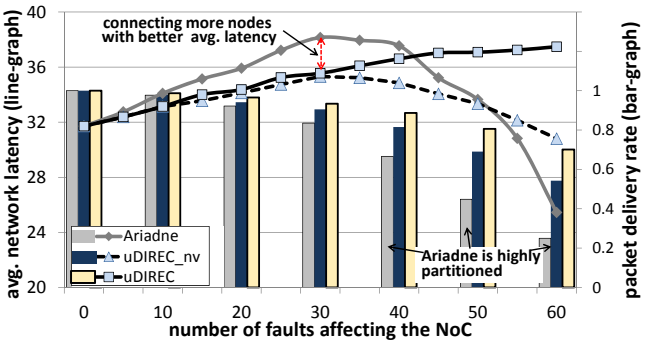


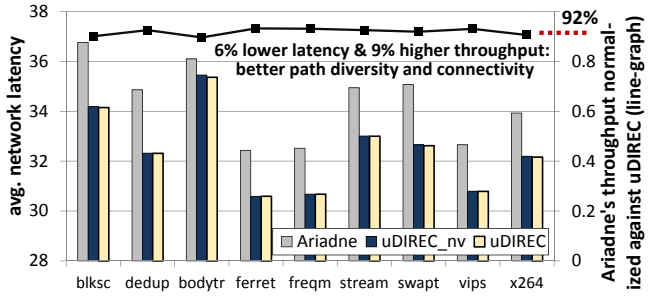
Figure 13: **Zero load latency**. Initially, latency degrades more gracefully for uDIREC as it provides path diversity. Beyond 40 faults, Ariadne becomes highly partitioned and hence latency drops steeply. Packet delivery rate is much lower for Ariadne, confirming its excessive partitioning. uDIREC_nv offers lower latency compared to uDIREC, but it shows greater partitioning.

Ariadne, respectively [1]. Finally, uDIREC_nv exhibits lower latency compared to uDIREC throughout the fault range, but it also suffers from greater partitioning. Still, the NoC partitioning with uDIREC_nv is not as severe as in the case of Ariadne.

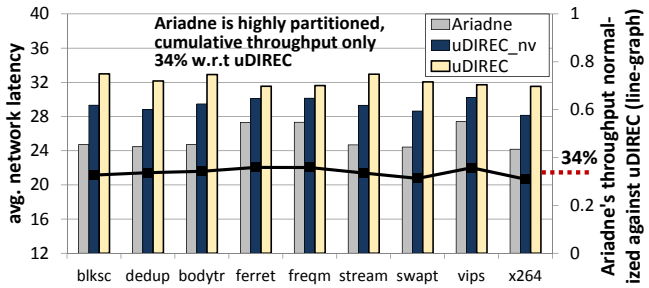
Similar trends are observed in simulations using PARSEC benchmark traces, as plotted in Figure 14. uDIREC's ability to provide path diversity leads to lower latencies at few faults (Figure 14a). At 30 faults, uDIREC shows 5.7% lower latency when compared to Ariadne, while delivering 8.8% higher cumulative throughput. At even higher faults, uDIREC loses nodes gracefully, and hence its latency decrease is not as rapid as Ariadne (Figure 14b). Note that at 60 faults, Ariadne's cumulative throughput is only 34% of uDIREC.

In addition to latency, we also measured saturation throughput of the largest surviving sub-network. Saturation throughput is the measure of maximum sustained bandwidth provided by the network and it is measured in flits received at each cycle. Figure 15 plots the packet throughput delivered by the network. uDIREC consistently delivers more packets per cycle as it uses additional unidirectional links to connect more nodes and enables more routes. uDIREC delivers 25% more packets per cycle than Ariadne at 15 faults. This advantage further increases to 39% at 60 faults. While not performing as well as uDIREC, uDIREC_nv still has 11% higher throughput compared to Ariadne at 15 faults. Again, both Immunet and Vicis can sustain a considerably lower maximum throughput compared to Ariadne [1], and hence their performance is even worse against uDIREC.

Performance, Energy and Fault-Tolerance (PEF) Metric [26]. Traditional NoC metrics, such as energy-delay product (EDP), do not capture the importance of reliability and its relation to both performance and power. To this end, [26] proposed a composite metric which unifies all three components: latency, energy, and fault-tolerance. They defined PEF as in Equation 1. In a fault-free network *Packet Delivery Rate*=1; thus, PEF becomes equal to EDP. We assume total network energy to be proportional to the number of active routers in the network, as we use identical routers in our setup. Therefore, we estimate per packet energy to be proportional to the fraction of number of active routers by packet throughput, as shown in Equation 2. Figure 16 shows PEF values for uDIREC variants normalized against PEF values for Ariadne. Note that a *lower* value



(a) **Low number (30) of faults**.



(b) **High number (60) of faults.**

Figure 14: **Average network latency with PARSEC benchmark traces**. The figure also shows cumulative throughput delivered by Ariadne (normalized against uDIREC). At few faults, uDIREC latency is lower compared to Ariadne as it provides additional path diversity, even though Ariadne delivers 8% fewer packets. At more faults, uDIREC connects significantly more nodes, with Ariadne delivering only 30% throughput compared to uDIREC. Therefore, packet latency with uDIREC is larger (packets traverse longer routes to destinations).

of PEF is *better*. The relative difference in the PEF values between uDIREC variants and Ariadne monotonically increases with the increasing number of faults. At 15 faults, uDIREC (uDIREC_nv) has 24% (17%) lower PEF than Ariadne, while showing more than 2× (1.6×) improvement at 60 faults. The reported PEF values confirm the benefits of using uDIREC across a wide range of fault rates. At few faults, the additional paths provided by uDIREC lead to reduced latency, while at higher number of faults, uDIREC delivers a greater fraction of the packets to their intended destinations.

$$PEF = \frac{(Average\ Latency) \times (Energy\ per\ Packet)}{Packet\ Delivery\ Rate} \quad (1)$$

$$Energy\ Per\ Packet \propto \frac{Number\ of\ Active\ Routers}{Packet\ Throughput} \quad (2)$$

6. RELATED WORK

Ensuring reliability in NoCs has been the subject of much previous research, focusing on a variety of aspects. Works such as [32, 36] focus on NoC protection against soft faults. Other methods enhance NoC reliability against permanent faults by enabling one or a combination of the following features: i) detection of erroneous behavior [20, 32], ii) diagnosis of fault site [12, 17, 20, 27, 42], iii) recovery from erroneous state [32], iv) system reconfiguration to bypass the permanent faults [1, 8, 17, 28, 41] or v) architectural protection for router logic [11,17,26]. Note that uDIREC is orthogonal to architectural approaches that extend the lifetime of NoC links

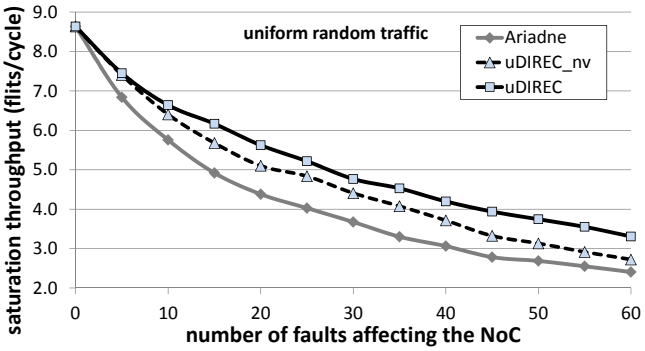


Figure 15: **Saturation throughput**. uDIREC consistently delivers more packets per cycle as it uses additional unidirectional links to connect more nodes and enable more routes.

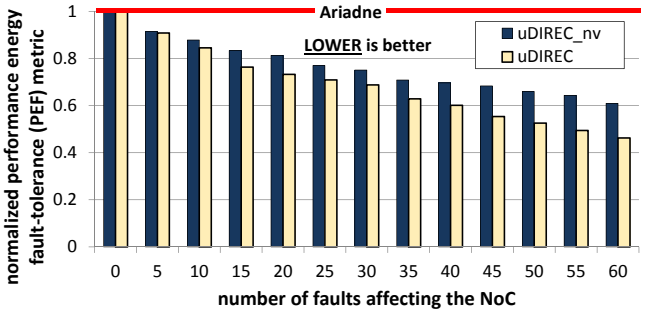


Figure 16: **Performance-energy-fault tolerance (PEF) metric [26]**. uDIREC monotonically improves with increasing faults. uDIREC (uDIREC_nv) shows 2× (1.6×) improvement at 60 faults.

(ECC [32], reversible transmission [3], partially-faulty links [34]) or components ([11, 26]). When the number of faults affecting a link/component are beyond repair using such approaches, the corresponding link/component can be switched off, and traffic re-routed around it using uDIREC. We also differtiate ourselves with adaptive routing algorithms, such as [23], that utilize escape VCs to overcome a few faults in regular topologies. Certain other routing algorithms [6, 27] flood or deflect packets to random neighbors to provide probabilistic reliability. In this work, we specifically investigate fine-resolution diagnosis and route-reconfiguration to cope with permanent faults.

During route-reconfiguration, a new set of deadlock-free routes are generated to replace the current ones, whenever a new fault is detected. The unpredictable number and location of fault occurrences, result in reconfiguration solutions designed for a bounded number [22, 23] or constrained pattern [9, 18, 19] of faults being unfit for NoCs. Therefore, we only compare against solutions that put no constraints on number and location of faults. Table 2 presents a qualitative comparison of the algorithms in this domain. All previous reconfiguration algorithms, either off-chip [31, 45, 46] or on-chip [1, 8, 17, 41], are limited to the granularity of a bidirectional link, if not any coarser. Therefore, they are limited by the shortcomings of the *Coarse_FM*, failing to capitalize on performance and reliability benefits of using the *Fine_FM*. This is confirmed by the fact that uDIREC drops less than $1/3^{rd}$ of the nodes when compared to the best performing prior-art [1].

Except for Vicis [17], which uses a costly BIST (10% overhead [16]) unit for diagnosis, no other previous solution presented a

solution	diagnosis support	resolution		node-drop rate	reconf area(%)
		diagnosis	reconf		
off-chip	NO	–	bi-link	high, >3×	23
Immunet	NO	–	bi-link	high, >3×	6
Vicis	**YES**	sgmt-pair	bi-link	high,dlock	1.5
Ariadne	NO	–	bi-link	high, >3×	2
uDIREC	**YES**	**segment**	**u-link**	**low, 1×**	**<1**

Table 2: **uDIREC's comparison with other reconfiguration solutions.** uDirec provides unified fault diagnosis and reconfiguration at fine granularity, resulting in better reliability characteristics. Moreover, the hardware structures required for uDIREC, are small and simple. The area numbers for schemes other than uDIREC are as reported in prior-work [1].

unified approach to diagnosis and reconfiguration. Typically, standalone route-reconfiguration schemes assume an ideal accuracy diagnosis scheme, which either localizes a fault to an entire link/router [31, 41, 45, 46], or to a *datapath segment pair* (defined in Section 2.1) [1].

uDIREC uses simple hardware additions to assist its software-based reconfiguration, incurring the lowest area cost among the route-reconfiguration schemes. Implementing off-chip reconfiguration schemes requires dedicated reliable resources for the collection of the surviving topology and the distribution of routing tables, to and from a central node, respectively. Ariadne [1] reports that the software-managed reconfiguration algorithms for off-chip networks, lead to 23.2% area overhead, if implemented on-chip without any modifications. Immunet [41], on the other hand, ensures reliable deadlock-free routes by reserving an escape virtual network with large buffers, which, in the worst case, reconfigures to form a unidirectional ring of surviving nodes. Vicis [17] leverages a bulky BIST unit at each router. In addition, the routing algorithm Vicis implements is not deadlock-free, and it often deadlocks at high number of faults. Finally, Ariadne [1] brings down the area overhead of reconfiguration by using a single wire to broadcast connection messages. However, this approach requires a coherent clock domain across the entire chip and presents a significant design challenge due to the complexity of its reconfiguration process.

7. CONCLUSIONS

We have presented uDIREC, a solution for reliable operation of NoCs providing graceful performance degradation even at large number of faults. uDIREC leverages a novel deadlock-free routing algorithm to maximally utilize all the working links in the NoC. Moreover, uDIREC incorporates a software-based fault diagnosis and reconfiguration algorithm that places no restriction on topology, router architecture or the number and location of faults. Simulations show that for a 64-node NoC at 15 faults, uDIREC drops 68% fewer nodes and provides 25% higher bandwidth over state-of-the-art reliability solutions. A combined performance, energy and fault-tolerance metric, that integrates energy-delay product with packet delivery rate, indicates 24% improvement at 15 faults, which more than doubles beyond 50 faults, showing that uDIREC is beneficial over a wide range of fault rates.

8. ACKNOWLEDGEMENTS

This work was supported by STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA, and NSF grant #0746425.

9. REFERENCES

[1] K. Aisopos, A. DeOrio, L.-S. Peh, and V. Bertacco. ARIADNE: Agnostic reconfiguration in a disconnected network environment. In *Proc. PACT*, 2011.

[2] K. Aisopos and L.-S. Peh. A systematic methodology to develop resilient cache coherence protocols. In *Proc. MICRO*, 2011.

[3] M. Al Faruque, T. Ebi, and J. Henkel. Configurable links for runtime adaptive on-chip communication. In *Proc. DATE*, 2009.

[4] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC benchmark suite: Characterization and architectural implications. In *Proc. PACT*, October 2008.

[5] T. Bjerregaard and S. Mahadevan. A survey of research and practices of network-on-chip. *ACM Computing Surveys*, 38(1), 2006.

[6] P. Bogdan, T. Dumitras, and R. Marculescu. Stochastic communication: A new paradigm for fault-tolerant networks-on-chip. *VLSI Design*, 2007.

[7] S. Borkar. Designing reliable systems from unreliable components: The challenges of transistor variability and degradation. *Micro, IEEE*, 25(6), 2005.

[8] F. Chaix, D. Avresky, N.-E. Zergainoh, and M. Nicolaidis. A fault-tolerant deadlock-free adaptive routing for on chip interconnects. In *Proc. DATE*, 2011.

[9] S. Chalasani and R. Boppana. Communication in multicomputers with nonconvex faults. *IEEE Trans. Computers*, 46(5), 1997.

[10] G.-M. Chiu. The odd-even turn model for adaptive routing. *IEEE Trans. Parallel and Distributed Systems*, 11(7), 2000.

[11] K. Constantinides, S. Plaza, J. Blome, B. Zhang, V. Bertacco, S. Mahlke, T. Austin, and M. Orshansky. BulletProof: A defect-tolerant CMP switch architecture. In *Proc. HPCA*, 2006.

[12] E. Cota, F. Kastensmidt, M. Cassel, M. Herve, P. Almeida, P. Meirelles, A. Amory, and M. Lubaszewski. A high-fault-coverage approach for the test of data, control and handshake interconnects in mesh networks-on-chip. *IEEE Trans. Computers*, 57(9), 2008.

[13] W. Dally and C. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Trans. Computers*, C-36(5), 1987.

[14] W. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., 2003.

[15] A. DeOrio, K. Aisopos, V. Bertacco, and L.-S. Peh. DRAIN: Distributed recovery architecture for inaccessible nodes in multi-core chips. In *Proc. DAC*, 2011.

[16] A. DeOrio, D. Fick, V. Bertacco, D. Sylvester, D. Blaauw, J. Hu, and G. Chen. A reliable routing architecture and algorithm for nocs. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 2012.

[17] D. Fick, A. DeOrio, J. Hu, V. Bertacco, D. Blaauw, and D. Sylvester. Vicis: A reliable network for unreliable silicon. In *Proc. DAC*, 2009.

[18] J. Flich, A. Mejia, P. Lopez, and J. Duato. Region-based routing: An efficient routing mechanism to tackle unreliable hardware in network on chips. In *Proc. NoCs*, 2007.

[19] Y. Fukushima, M. Fukushi, and S. Horiguchi. Fault-tolerant routing algorithm for network on chip without virtual channels. In *Proc. DFT*, 2009.

[20] A. Ghofrani, R. Parikh, A. Shamshiri, A. DeOrio, K.-T. Cheng, and V. Bertacco. Comprehensive online defect diagnosis in on-chip networks. In *Proc. VTS*, 2012.

[21] C. Glass and L. Ni. The turn model for adaptive routing. In *Proc. ISCA*, 1992.

[22] C. J. Glass and L. M. Ni. Fault-tolerant wormhole routing in meshes without virtual channels. *IEEE Trans. Parallel and Distributed Systems*, 7, 1996.

[23] M. Gomez, J. Duato, J. Flich, P. Lopez, A. Robles, N. Nordbotten, O. Lysne, and T. Skeie. An efficient fault-tolerant routing methodology for meshes and tori. *Comp. Arch. Letters*, 3(1), 2004.

[24] S. Gupta, S. Feng, A. Ansari, J. Blome, and S. Mahlke. The StageNet fabric for constructing resilient multicore systems. In *Proc. MICRO*, 2008.

[25] A. Kahng, B. Li, L.-S. Peh, and K. Samadi. Orion 2.0: A fast and accurate NoC power and area model for early-stage design space exploration. In *Proc. DATE*, 2009.

[26] J. Kim, C. Nicopoulos, D. Park, V. Narayanan, M. Yousif, and C. Das. A gracefully degrading and energy-efficient modular router architecture for on-chip networks. In *Proc. ISCA*, 2006.

[27] A. Kohler and M. Radetzki. Fault-tolerant architecture and deflection routing for degradable noc switches. In *Proc. NoCs*, 2009.

[28] M. Koibuchi, H. Matsutani, H. Amano, and T. M. Pinkston. A lightweight fault-tolerant mechanism for network-on-chip. In *Proc. NoCs*, 2008.

[29] O. Lysne, J. M. Montañana, J. Flich, J. Duato, T. M. Pinkston, and T. Skeie. An efficient and deadlock-free network reconfiguration protocol. *IEEE Trans. Computers*, 57(6), 2008.

[30] M. Martin, D. Sorin, B. Beckmann, M. Marty, M. Xu, A. Alameldeen, K. Moore, M. Hill, and D. Wood. Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset. *ACM SIGARCH Computer Architecture News*, 33(4), 2005.

[31] A. Mejia, J. Flich, J. Duato, S.-A. Reinemo, and T. Skeie. Segment-based routing: An efficient fault-tolerant routing algorithm for meshes and tori. In *Proc. IPDPS*, 2006.

[32] S. Murali, T. Theocharides, N. Vijaykrishnan, M. Irwin, L. Benini, and G. De Micheli. Analysis of error recovery schemes for networks on chips. *IEEE Design & Test*, 22(5), 2005.

[33] E. B. Nightingale, J. R. Douceur, and V. Orgovan. Cycles, cells and platters: An empirical analysis of hardware failures on a million consumer PCs. In *Proc. EUROSYS*, 2011.

[34] M. Palesi, S. Kumar, and V. Catania. Leveraging partially faulty links usage for enhancing yield and performance in networks-on-chip. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 29(3), 2010.

[35] R. Parikh and V. Bertacco. Formally enhanced verification at runtime to ensure noc functional correctness. In *Proc. MICRO*, 2011.

[36] D. Park, C. Nicopoulos, J. Kim, N. Vijaykrishnan, and C. R. Das. Exploring fault-tolerant network-on-chip architectures. In *Proc. DSN*, 2006.

[37] A. Pellegrini, J. L. Greathouse, and V. Bertacco. Viper: virtual pipelines for enhanced reliability. In *Proc. ISCA*, 2012.

[38] T. M. Pinkston, R. Pang, and J. Duato. Deadlock-free dynamic reconfiguration schemes for increased network dependability. *IEEE Trans. Parallel and Distributed Systems*, 14(8), 2003.

[39] A. Prodromou, A. Panteli, C. Nicopoulos, and Y. Sazeides. Nocalert: An on-line and real-time fault detection mechanism for network-on-chip architectures. In *Proc. MICRO*, 2012.

[40] M. Prvulovic, Z. Zhang, and J. Torrellas. Revive: cost-effective architectural support for rollback recovery in shared-memory multiprocessors. In *Proc. ISCA*, 2002.

[41] V. Puente, J. A. Gregorio, F. Vallejo, and R. Beivide. Immunet: A cheap and robust fault-tolerant packet routing mechanism. In *Proc. ISCA*, 2004.

[42] J. Raik, R. Ubar, and V. Govind. Test configurations for diagnosing faulty links in noc switches. In *Proc. ETS*, 2007.

[43] V. Reddi and D. Brooks. Resilient architectures via collaborative design: Maximizing commodity processor performance in the presence of variations. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 30(10), 2011.

[44] J. Sancho, A. Robles, and J. Duato. An effective methodology to improve the performance of the up*/down* routing algorithm. *IEEE Trans. Parallel and Distributed Systems*, 15(8), 2004.

[45] J. C. Sancho, A. Robles, and J. Duato. A flexible routing scheme for networks of workstations. In *Proc. ISHPC*, 2000.

[46] M. Schroeder, A. Birrell, M. Burrows, H. Murray, R. Needham, T. Rodeheffer, E. Satterthwaite, and C. Thacker. Autonet: A high-speed, self-configuring local area network using point-to-point links. *IEEE Trans. Selected Areas in Communication*, 9(8), 1991.

[47] S. Shamshiri, A. Ghofrani, and K.-T. Cheng. End-to-end error correction and online diagnosis for on-chip networks. In *Proc. ITC*, 2011.

[48] J. Srinivasan, S. Adve, P. Bose, and J. Rivers. The impact of technology scaling on lifetime reliability. In *Proc. DSN*, 2004.

[49] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar, and S. Borkar. An 80-tile sub-100-w teraflops processor in 65-nm cmos. *IEEE Journal of Solid-State Circuits*, 2008.

[50] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. Brown, and A. Agarwal. On-chip interconnection architecture of the tile processor. *Micro, IEEE*, 27(5), 2007.